Applications of meta-heuristics to traffic engineering in IP networks

Bernard Fortz *

Université Libre de Bruxelles, Faculté des Sciences, Département d'Informatique
and CORE, Université catholique de Louvain
Email: bernard.fortz@ulb.ac.be

Abstract

Intra-domain routing protocols are based on Shortest Path First (SPF) routing, where shortest paths are calculated between each pair of nodes (routers) using pre-assigned link weights, also referred to as *link metric*. These link weights can be modified by network administrators in accordance with the routing policies of the network operator. The operator's objective is usually to minimize traffic congestion or minimize total routing cost subject to the traffic demands and the protocol constraints. However, determining a link weights combination that best suits the network operator's requirements is a difficult task.

This paper provides a survey of meta-heuristic approaches to traffic engineering, focusing on local search approaches and extensions to the basic problem taking into account changing demands and robustness issues with respect to network failures.

1 Introduction

Provisioning an Internet Service Provider (ISP) backbone network for intra-domain IP traffic is a big challenge, particularly due to rapid growth of the network and user demands. At times, the network topology and capacity may seem insufficient to meet the current demands. At the same time, there is mounting pressure for ISPs to provide Quality of Service (QoS) in terms of Service Level Agreements (SLAs) with customers, with loose guarantees on delay, loss, and throughput. All of these issues point to the importance of *traffic engineering*, making more efficient use of existing network resources by tailoring routes to the prevailing traffic.

1.1 The general routing problem

Optimizing the use of existing network resources can be seen as a general routing problem defined as follows. We are given a directed network $G = (N, A)$ with a capacity c_a for each $a \in A$, and a demand matrix D that, for each pair $(s, t) \in N \times N$, tells the demand $D(s, t)$ in traffic flow between s and t. We sometimes refer to the non-zero entries of D as the *demands*. The set of arcs leaving a node u is denoted by $\delta^+(u) := \{(u, v) : (u, v) \in A\}$ while the set of arcs entering a node u is denoted by $\delta^-(u) := \{(v, u) : (v, u) \in A\}$

*The author has been supported by the Communauté française de Belgique - Actions de Recherche Concertées (ARC).

With each arc $a \in A$, we associate a cost function $\Phi_a(l_a)$ of the load l_a, depending on how close the load is to the capacity c_a. We assume in the following that Φ_a is an strictly increasing and convex function. Our formal objective is to distribute the demanded flow so as to minimize the sum

$$\Phi = \sum_{a \in A} \Phi_a(l_a)$$

of the resulting costs over all arcs. Usually, Φ_a increases rapidly as loads exceeds capacities, and our objective typically implies that we keep the max-utilization $\max_{a \in A} l_a/c_a$ below 1, or at least below 1.1, if at all possible.

In this general routing problem, there are no limitations to how we can distribute the flow between the paths. With each pair $(s,t) \in N \times N$ and each arc $a \in A$, we associate a variable $f_a^{(s,t)}$ telling how much of the traffic flow from s to t goes over a. Moreover, for each arc $a \in A$, variable l_a represents the total load on arc a, i.e. the sum of the flows going over a. With these notation, the problem can be formulated as the following multi-commodity flow problem.

$$\min \Phi = \sum_{a \in A} \Phi_a(l_a)$$

subject to

$$\sum_{a \in \delta^+(u)} f_a^{(s,t)} - \sum_{a \in \delta^-(u)} f_a^{(s,t)} = \begin{cases} D(s,t) & \text{if } u=s, \\ -D(s,t) & \text{if } u=t, \\ 0 & \text{otherwise,} \end{cases} \qquad u,s,t \in N, \tag{1}$$

$$l_a = \sum_{(s,t) \in N \times N} f_a^{(s,t)} \qquad a \in A, \tag{2}$$

$$f_a^{(s,t)} \geq 0 \qquad a \in A;\ s,t \in N. \tag{3}$$

Constraints (1) are flow conservation constraints that ensure the desired traffic flow is routed from s to t, and constraints (2) define the load on each arc.

As Φ is a convex objective function and all constraints are linear, this problem can be solved optimally in polynomial time. We denote by Φ_{OPT} the optimal solution of this general routing problem.

In our experiments, Φ_a are piecewise linear functions, with $\Phi_a(0) = 0$ and derivative

$$\Phi_a'(l) = \begin{cases} 1 & \text{for} \quad 0 \leq l/c_a < 1/3, \\ 3 & \text{for} \quad 1/3 \leq l/c_a < 2/3, \\ 10 & \text{for} \quad 2/3 \leq l/c_a < 9/10, \\ 70 & \text{for} \quad 9/10 \leq l/c_a < 1, \\ 500 & \text{for} \quad 1 \leq l/c_a < 11/10, \\ 5000 & \text{for} \quad 11/10 \leq l/c_a < \infty. \end{cases} \tag{4}$$

The function Φ_a is illustrated in Figure 1, and can be viewed as modeling retransmission delays caused by packet losses. Generally it is cheap to send flow over an arc with a small utilization l_a/c_a. The cost increases progressively as the utilization approaches 100%, and explodes when we go above 110%. With this cost function, the general routing problem becomes a linear program.

The objective function was chosen on the basis of discussions on costs with people close to the AT&T IP backbone. Motivations on our choice for the objective function and the different model assumptions are discussed in detail in [17], and, for a closely related application, in [15]. A description of the general infrastructure behind this kind of traffic engineering is given in [13].

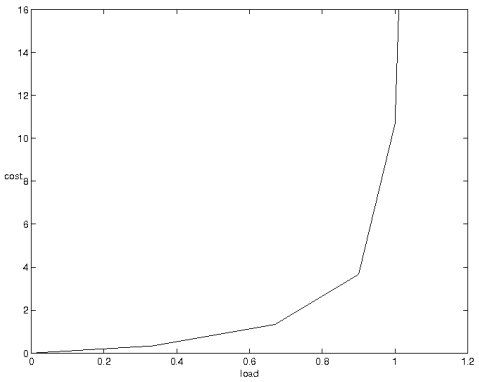

Figure 1: Arc cost $\Phi_a(l_a)$ as a function of load l_a for arc capacity $c_a = 1$.

1.2 The OSPF weight setting problem

The most commonly used intra-domain internet routing protocols today are shortest path protocols such as Open Shortest Path First (OSPF) [22]. OSPF does not support a free distribution of flow between source and destination as defined above in the general routing problem. In OSPF, the network operator assigns a weight w_a to each link $a \in A$, and shortest paths from each router to each destination are computed using these weights as lengths of the links. In practice, link weights are integer encoded on 16 bits, therefore they can take any value between 1 and 65,535. In each router, represented by a node of the graph, the next link on all shortest paths to all possible destinations is stored in a table. A flow arriving at the router is sent to its destination by splitting the flow between the links that are on the shortest paths to the destination. The splitting is done using pseudo-random methods leading to an approximately even splitting. For simplicity, we assume that the splitting is exactly even (for AT&T's WorldNet this simplification leads to reasonable estimates). This splitting rule is usually called Equal Cost Multi-Path (ECMP).

More precisely, given a set of weights $(w_a)_{a \in A}$, the length of a path is then the sum of its arc weights, and we have the extra condition that all flow leaving a node aimed at a given destination is evenly spread over the first arcs on shortest paths to that destination. Therefore, for each source-destination pair $(s,t) \in N \times N$ and for each arc $a \in \delta^+(u)$ for some node $u \in N$, we have that $f_a^{(s,t)} = 0$ if a is not on a shortest path from s to t, and that $f_a^{(s,t)} = f_{a'}^{(s,t)}$ if both $a \in \delta^+(u)$ and $a' \in \delta^+(u)$ are on shortest paths from s to t. Note that the routing of the demands is completely determined by the shortest paths which in turn are determined by the weights we assign to the arcs.

The quality of OSPF routing depends highly on the choice of weights. Nevertheless, as recommended by Cisco (a major router vendor) [20], these are often just set inversely proportional to the capacities of the links, without taking any knowledge of the demand into account.

The *OSPF weight setting problem* is to set the weights so as to minimize the cost of the resulting routing.

This paper provides a survey of meta-heuristics techniques for the OSPF weight setting problem and some of its extensions. Another recent survey by Altin et al. [2] also discusses in more details the origins of the problem, the possible choices of objective functions, and exact methods for the problem. In Section 2, we survey meta-heuristics that have been proposed to solve the basic version of the problem. Section 3 presents extensions taking into account changing demands and robustness issues with respect to network failures.

2 Survey of meta-heuristics

Shortest path routing problems are NP-hard. Direct formulations are extremely hard to solve, and integer programming approaches can typically solve only small to medium size problems. Moreover, in an operational setting, additional constraints can appear that are difficult to integrate in a mixed-integer programming formulations. Therefore, for large networks instances, heuristics can be necessary to find good feasible solutions in a limited computing time. In Section 2.1, we present a local search approach to the problem. Section 2.2 examines further some issues that need to be considered in order to obtain an effective heuristic, and Section 2.3 presents a method for finding good starting solutions. In Section 2.4, other approaches are briefly discussed.

2.1 A local search heuristic

One of the first heuristic approaches to the OSPF weight setting problem is a local search approach developed by Fortz and Thorup [14, 17]. Recently, a similar implementation has been made available in the opensource TOTEM toolbox [21]. The local search algorithm is called IGP-WO in TOTEM.

In OSPF routing, for each arc $a \in A$, we have to choose a weight w_a. These weights uniquely determine the shortest paths, the routing of traffic flow, the loads on the arcs, and finally, the value of the cost function Φ.

Suppose that we want to minimize a function f over a set X of feasible solutions. Local search techniques are iterative procedures that for each iteration define a neighborhood $\mathcal{N}(x) \subseteq X$ for the current solution $x \in X$, and then choose the next solution x' from this neighborhood. Often we want the neighbor $x' \in \mathcal{N}(x)$ to improve on f in the sense that $f(x') < f(x)$.

In the remainder of this section, we first describe the neighborhood structure we apply to solve the weight setting problem. Second, using hashing tables, we address the problem of avoiding cycling. These hashing tables are also used to avoid repetitions in the neighborhood exploration. While the neighborhood search aims at intensifying the search in a promising region, it is often of great practical importance to search a new region when the neighborhood search fails to improve the best solution for a while. These techniques are called search diversification. We refer the reader to [17] for a description of the diversification techniques we use.

A solution of the weight setting problem is completely characterized by its vector $w = (w_a)_{a \in A}$ of weights, where $w_a \in W$, the set of possible weights. We define a neighbor $w' \in \mathcal{N}(w)$ of w by one of the two following operations applied to w.

Single weight change. This simple modification consists in changing a single weight in w. We define a neighbor w' of w for each arc $a \in A$ and for each possible weight $t \in W \backslash \{w_a\}$ by setting $w'(a) = t$ and $w'(b) = w_b$ for all $b \neq a$.

Evenly balancing flows. Assuming that the cost function Φ_a for an arc $a \in A$ is increasing and convex, meaning that we want to avoid highly congested arcs, we want to split the flow as evenly as possible between different arcs.

More precisely, consider a demand node t such that $\sum_{s \in N} D(s,t) > 0$ and some part of the demand going to t goes through a given node u. Intuitively, we would like OSPF routing to split the flow to t going through u evenly along arcs leaving u. This is the case if every arc in $\delta^+(u)$ belongs to a shortest path from u to t. More precisely, if $\delta^+(u) = \{a_i : 1 \leq i \leq p\}$, and if P_i is one of the shortest paths from the tail of a_i to t, for $i = 1, \ldots, p$, then we want to set w' such that

$$w'_{a_i} + w'(P_i) = w'_{a_j} + w'(P_j) \qquad 1 \leq i, j \leq p,$$

where $w'(P_i)$ denotes the sum of the weights of the arcs belonging to P_i. A simple way of achieving this goal is to set

$$w'(a) = \begin{cases} w^* - w(P_i) & \text{if } a = a_i, \text{ for } i = 1, \ldots, p, \\ w_a & \text{otherwise.} \end{cases}$$

where $w^* = 1 + \max_{i=1,\ldots,p}\{w(P_i)\}$.

A drawback of this approach is that an arc that does not belong to one of the shortest paths from u to t may already be congested, and the modifications of weights we propose will send more flow on this congested arc, an obviously undesirable feature. We therefore decided to choose at random a threshold ratio θ between 0.25 and 1, and we only modify weights for arcs in the maximal subset B of $\delta^+(u)$ such that

$$w_{a_i} + w(P_i) \leq w_{a_j} + w(P_j) \quad \forall i : a_i \in B, \; j : a_j \notin B,$$
$$l_a^w \leq \theta \; c_a \quad \forall a \in B,$$

where l_a^w denotes the load on a resulting from weight vector w. The last relation implies that the utilization of an arc $a \in B$ resulting from the weight vector w is less than or equal to θ, so that we can avoid sending flow on already congested arcs. In this way, flow leaving u towards t can only change for arcs in B, and choosing θ at random allows to diversify the search.

This choice of B does not ensure that weights remain below w_{max}. This can be done by adding the condition $\max_{i:a_i \in B} w(P_i) - \min_{i:a_i \in B} w(P_i) \leq w_{max}$ when choosing B.

The simplest local search heuristic is the descent method that, at each iteration, selects the best element in the neighborhood and stops when this element does not improve the objective function. This approach leads to a local minimum that is often far from the optimal solution of the problem, and heuristics allowing non-improving moves have been considered. Unfortunately, non-improving moves can lead to cycling, and one must provide mechanisms to avoid it.

Our choice was to use hashing: hash functions compress solutions into single integer values, sending different solutions into the same integer with small probability. We use a boolean table T to record if a value produced by the hash function $h()$ has been encountered. At the beginning of the algorithm, all entries in T are set to false. If w is the solution produced at a given iteration, we set $T(h(w))$ to true, and, while searching the neighborhood, we reject any solution w' such that $T(h(w'))$ is true. Checking that a solution has been encountered is therefore performed in constant time. The hash function we used is described in [17].

2.2 Effectiveness issues

To evaluate the cost of a solution represented as a set of weights, we have to compute the shortest paths for all origin-destination pairs, then to send the flows along the shortest paths according to the ECMP splitting rule. This could be a bottleneck in the search for good solutions as computing this cost function from scratch is computationally expensive.

To overcome that difficulty, we can apply a fast algorithmic approach to compute the flows. We use a two-step algorithm based on the shortest path computation. In the first step we compute all the shortest paths with respect to the current weights for all node pairs, and then, in the second step, we recursively assign flows to the paths computed in the first phase (see [17] or Algorithm 7.1 in [25]).

In most heuristic approaches, the number of changes in the shortest paths graph and in the flows is very small between neighboring solutions. Hence, using fast updates of shortest paths and flows is crucial to make heuristics effective. We now briefly review these approaches.

With respect to shortest paths, this idea is already well studied by Ramalingam and Reps [26], and we can apply their algorithm directly. Their basic result is that, for the re-computation, we only spend

time proportional to the number of arcs incident to nodes s whose distance to t changes. In typical experiments there were only very few changes, so the gain is substantial - in the order of factor 15 for a 100 node graph. An improved algorithm was recently proposed by Buriol et al [11].

To update the flows, a similar approach, described in [17], can be used. Experiments reported in that paper show that using dynamic updates of shortest paths and flows make the algorithm from 5 up to 25 times faster, with an average of 15 times faster.

2.3 Improvements using column generation

The general routing problem introduced in Section 1.1 provides a good lower bound for the IGP weight setting problem in practice, as shown in [17]. This general routing problem can also be formulated with path flow variables, and solved with column generation.

The main advantage of this approach is that dual variables used in the pricing problem are natural candidates as link weights. Fortz and Ümit [18] managed to significantly improve the results obtained by the heuristic by warm-starting the local search with the best solution obtained with dual variables of the multi-commodity flow relaxation of the problem.

2.4 Other heuristic approaches

Ericsson et al [12] have proposed a genetic algorithm for the same problem. Solutions are naturally represented as vectors of weights, and the crossover procedure used is *random keys*, that was first proposed by Bean [5]. To cross and combine two parent solutions p_1 (elite) and p_2 (non-elite), first generate a random vector r of real numbers between 0 and 1. Let K be a cutoff real number between 0.5 and 1, which will determine if a gene is inherited from p_1 or p_2. A child c is generated as follows: for all genes i, if $r[i] < K$, set $c[i] = p_1[i]$ otherwise set $c[i] = p_2[i]$. They also implemented a mutation operator that randomly mutates a single weight.

This approach was improved by Buriol et al [10]. They added a local search procedure after the crossover to improve the population. This hybrid approach, combined with dynamic updates of shortest paths and flows, lead to results competitive in quality to the local search of Fortz and Thorup, with a slightly faster convergence. Another application of genetic algorithms to Shortest Path Routing (SPR) design can be found in [23].

A simulated annealing approach was proposed by Ben-Ameur [7] for the single path routing case. Another line of heuristic approaches comes from using Lagrangean relaxations of the mixed integer programming models (see e.g. [9] for single path routing and [19] for ECMP routing).

3 Extensions

The basic traffic engineering problem is very important, but usually does not meet the requirements of network operators as changing demands or network failures are not taken into account. We now discuss some extensions to the basic problem. In Section 3.1, we consider fast re-optimization with a few weight changes. Then, in Section 3.2 we consider the simultaneous optimization over multiple demand matrices. This comes as a building block for heuristics dealing with failure scenarios (Section 3.3) as well as for polyhedral demand uncertainty (Section 3.4).

3.1 Re-optimization with few weight changes

The problem of optimizing OSPF weights was mostly studied with a given *fixed* set of demands and a *fixed* network topology.

However, demand matrices and networks *change*. In case of changes that degrade performance too much, most network operators do not like to make many weight changes, for two basic reasons:

(i) Weights are typically not changed centrally, so changing a lot of weights may require substantial network management overhead, and creates the risk of a big protocol overhead.

(ii) Network operators are often not comfortable with many weight changes. There may be many aspects to good routing beyond the simple objectives presented in this paper. Therefore major weight changes are not welcomed when a first weight setting giving a satisfactory routing has been established.

Therefore, we want to make as few weight changes as possible. The local search from [17] works with a single solution that is iteratively improved by small changes in the current solution. It typically performs a lot of iterations (5000 in our practical experiments), and therefore produces a solution completely different from the starting one. This can be seen as a depth-first search in the solution space, but since we want to make as few changes as possible, our approach should rather be a breadth first search.

Another heuristic was proposed in [15] for improving an input weight setting w_0 with as few weight changes as possible. It works as follows: first we consider about 1000 single weight changes to w_0, corresponding to about five weight changes for each arc in our largest networks. The number of weight changes considered is limited by applying random sampling to the neighborhood structure, as exploring the full neighborhood is too time-consuming. Instead of selecting only the best weight change as in classical local search heuristics, we keep the 100 best weight changes in a family F of "best weight settings". The process is iterated with F instead of w_0: we consider 1000 single weight changes for each weight setting in F and a new F is selected containing the 100 best of the old weight settings in F. After i iterations, including the start from w_0, the family consists of weight settings with up to i weight changes from w_0. The size of F corresponds to the breadth of our search. All the above numbers are just parameters that experimentally were found to give a good compromise between quality of solution and time.

This technique for few changes has, to our knowledge, not been used before. Its main interest is that it provides a general framework for optimizing with few changes that can be easily adapted for other applications. It has the advantage that if a local search heuristic is available, the main ingredients such as the changes applied to one solution to get a neighbor of it, or the procedures to evaluate a new solution, can be reused in this new framework, saving a lot of implementation work.

3.2 Multiple demand matrices

Our motivation for working with multiple demand matrices is the general experience from AT&T that traffic follows quite regular periods with a peak in the day and in the evening. The network operators do not want to change weights on a regular basis so we want just one weight setting which is good for the whole period. We then collect a peak demand matrix for the day and one for the evening. A weight setting performing good on both performs good on all convex combinations, and hence it has a good chance of performing well for the whole period. This approach was also introduced in [15].

Given a network $G = (N, A, c)$ with several demand matrices $D_1, ..., D_k$, we want to find a single weight setting $w := (w_a)_{a \in A}$ which works well for all of them. In general, we will use $\ell_a(G, D, w)$ to

denote the load on link a with network G, demand matrix D, and weight setting w. Similarly for our cost function, we have $\Phi(G, D, w) = \sum_a \Phi_a(\ell_a(G, D, w))$ with Φ_a as defined in (4).

Now consider a demand matrix D dominated by a convex combination of $D_1, ..., D_k$, that is $D \leq \alpha_1 D_1 + \cdots + \alpha_k D_k$ where $\alpha_1 + \cdots + \alpha_k = 1$. Here everything is understood to be entry-wise, so for all x, y, $D[x, y] \leq \alpha_1 D_1[x, y] + \cdots + \alpha_k D_k[x, y]$.

Since the routing for each source-destination pair is fixed by the weight setting w, for each arc $a \in A$, $\ell_a(G, D, w) \leq \alpha_1 \ell_a(G, D_1, w) + \cdots + \alpha_k \ell_a(G, D_k, w)$. In particular, it follows that the max-utilization for D is no worse that the worst max-utilization for the D_i. Furthermore, since each arc cost function Φ_a is convex, $\Phi_a(\ell_a(G, D, w)) \leq \alpha_1 \Phi_a(\ell_a(G, D_1, w)) + \cdots + \alpha_k \Phi_a(\ell_a(G, D_k, w))$, and hence $\Phi(G, D, w) \leq \alpha_1 \Phi(G, D_1, w) + \cdots + \alpha_k \Phi(G, D_k, w)$. Thus, our weight setting w does no worse for D than for the worst of the D_i, neither with respect to our cost function Φ, nor with respect to max-utilization. Note that the same observation holds true with Multi-Protocol Label Switching (MPLS), as long as the routing for each source-destination pair is fixed.

To optimize simultaneously for several demand matrices $D_1, ..., D_k$, we simply modify the local search heuristic to minimize

$$\Phi(G, D_1, ..., D_k, w) = \sum_{i \leq k} \Phi(G, D_i, w) \tag{5}$$

As in our original motivation for defining Φ, this has the the effect of penalizing highly loaded links, this time, for all the demand matrices instead of just one.

3.3 Robust optimization for single link failures

The heuristic of Section 2.1 has been designed to provide a weight setting for a single demand matrix and a fixed network. We now consider the possibility of link failures. The approach below was proposed in [16]. We define a *state* of the network as a subset $S \subseteq A$ of arcs containing the arcs that are operational. The states considered here are the *normal state* A and the *single link failure states* $A \setminus \{a\}$ for each link $a \in A$. In SPF protocols, the network operator assigns a weight to each link, and shortest paths from each router to each destination are computed using these weights as lengths of the links. These shortest paths are updated each time the network state changes. Given a network state S, a weight setting w and a vector of arc capacities c, we denote by $\Phi(S, w, c)$ the cost of the routing obtained with weight setting w in state S of the network, using the piecewise linear cost function described in Section 1.1.

We suppose that once the operator has fixed the set of OSPF weights, he does not want to change it whatever the state of the network is. The possibility of getting a better routing in case of failures by allowing a few weight changes has been studied in [15].

The cost function we use has been designed in such a way that it tries to keep the flow on each link below the capacity of that link. This is an objective we want to maintain for each link failure. In the normal state, however, the operator usually wants the flows to remain much more below the capacity, in order to be more robust in cases of increasing demand and to ensure capacity will be available to perform the rerouting in case of failure. Let α be the maximal ratio of the capacity the network operator wants to use in the normal state. In our experiments, we assumed $\alpha = 0.6$. Therefore, in the normal state, the operator wants w to minimize $\Phi(A, w, \alpha c)$. We suppose here that the operator gives an equal importance to the quality of the routing in the normal state and to its robustness (i.e. the quality of the routing in all the single link failure states). Moreover, we assume all the link failures have the same importance, but it is trivial to extend our results by giving a weight to each link failure.

Putting it all together, we want to find a weight setting w^* that solves

$$\min_w \Psi(w) := \frac{1}{2} \left(\Phi(A, w, \alpha c) + \frac{1}{m} \sum_{a \in A} \Phi(A \setminus \{a\}, w, c) \right) \tag{6}$$

where c is the capacity vector and $m := |A|$ the number of links in the network. A brute force approach would be to use the heuristic presented in Section 2.1 to optimize $\Psi(w)$. However, this would require the evaluation of all the $m + 1$ scenarios for each weight setting encountered, therefore increasing the computing time by a factor m.

Our approach to reduce the computing time is the following. We hope that only a few link failures will be representative of "bad cases" and will contribute for a large part of the total cost in (6). At each iteration, we maintain a list of *critical links* $\mathcal{C}$ and only evaluate the cost function restricted to the corresponding states, i.e. we evaluate each weight setting w in the neighborhood of the current iterate with the cost function

$$\Psi(w, \mathcal{C}) := \frac{1}{2}\left(\Phi(A, w, \alpha c) + \frac{1}{|\mathcal{C}|}\sum_{a\in\mathcal{C}}\Phi(A \setminus \{a\}, w, c)\right).$$

The heuristic starts with $\mathcal{C} = \emptyset$. It is essentially the same as before, adapted to optimize $\Psi(w, \mathcal{C})$, with the addition that $\mathcal{C}$ is updated every T iterations. We update $\mathcal{C}$ as follows. Let $u(a, w)$ be the maximum utilization with weight setting w when arc a has failed, i.e.

$$u(a, w) = \max_{b\in A\setminus\{a\}} \frac{l(b, w, a)}{c_b},$$

where $l(b, w, a)$ denotes the load on arc b with weight setting w when a has failed, and let $\overline{u}$ be the average maximum utilization over all scenarios in the critical set, i.e. $\overline{u} := \frac{1}{|\mathcal{C}|}\sum_{a\in\mathcal{C}} u(a, w)$. We first choose the arc a not in the critical set that maximizes $u(a, w)$. If $u(a, w) > \overline{u}$, we add a to $\mathcal{C}$. Moreover, we want to keep $\mathcal{C}$ of small size. To this end, we fix a maximal size K and we remove the arc that minimizes $u(a, w)$ over $\mathcal{C}$ from the critical set each time its size exceeds K.

3.4 Polyhedral demand uncertainty

For a given network, the traditional routing problem deals with selecting paths to transfer a 'given' set of demands from their origins to destinations. In this general definition, there is no restriction on the structure of the paths to be used, and it is assumed that the amounts of traffic between all origin and destination pairs are already known. However, several restrictions are imposed on the path structure in telecommunication networks, and designing a reliable network using a single demand matrix strains credibility as the network size and the service variety increase in the contemporary business world. It is not likely to anticipate fluctuations in demand expectations without overestimations, which would lead to the waste of network resources or a high service cost. A well-known online approach to handle such shifts is to update routes adaptively as some changes are observed. However, the additional benefits of these methods are not for free since excessive modifications might ruin the consistency and dependability of network operations. At this point, off-line methods based on optimizing over a *set* of traffic matrices have started to win adherents [1, 4, 6, 8, 24].

The general method is to use either a discrete set and hence a scenario-based optimization or a polyhedral set defined by network characteristics. Then the motivation is to determine the routing whose worst case performance for any feasible realization in this set is the best. Such a routing is called *oblivious* since it is determined irrespective of a specific demand matrix. Applegate and Cohen [4] discuss the general routing problem with almost no information on traffic demands. Later, Belotti and Pinar [6] incorporate box model of uncertainty as well as statistical uncertainty into the same problem. Ben-Ameur and Kerivin [8] study the minimum cost general oblivious routing problem under polyhedral demand uncertainty and use an algorithm based on iterative path and constraint generation as a solution tool. Mulyana and Killat [24] deal with the OSPF routing problem, where traffic uncertainty is described by a set of outbound constraints. Finally, Altın et al. [1] study polyhedral demand uncertainty with OSPF routing under weight management and provide a compact Mixed Integer Programming (MIP) formulation and a branch-and-price algorithm.

In this section, we discuss oblivious OSPF routing with weight management and polyhedral demands, and describe the solution strategy proposed in Altın et al. [3]. Optimizing weights would enable traffic engineering with OSPF since link metric is the only tool we can employ to manipulate routes so as to make OSPF more comparable to other flexible protocols like MPLS. Moreover, polyhedral demands make the problem more practically defensible by ensuring a design that remains robust under a range of applicable shifts in traffic demand.

Let us relax the assumption of a fixed traffic matrix d and consider a polyhedron $\mathfrak{D}$ of feasible demand realizations. Now, the concern is to determine paths such that any $d \in \mathfrak{D}$ can be accommodated efficiently. This means that our 'optimal' routing will have the best worst case performance for $\mathfrak{D}$ independently of a specific traffic matrix. The main impact of such a shift will be a change in the definition of load on each link $a \in A$. It is now defined as a function of d, which can be any vector in $\mathfrak{D}$. Consequently, for the polyhedral case, the load definition becomes

$$l_a \geq \sum_{(s,t)\in Q} d_{st} f_a^{st} \quad d \in \mathfrak{D},\ a \in A \tag{7}$$

where $\mathfrak{D}$ is an arbitrary polyhedron. However, this change leads to a semi-infinite optimization problem. We can eliminate this difficulty by using a duality transformation. Details can be found in [3].

We now discuss our algorithmic approach to tackle polyhedral demands. It has two main steps, namely the traffic matrix enumeration and weight optimization. We use IGP-WO, the TOTEM weight optimizer, for the latter step, whereas we use a mixed integer programming formulation solved with CPLEX for the first part. As the representation theorem for polytopes suggests, any traffic matrix $d \in \mathfrak{D}$ can be represented as a convex combination of the extreme points of $\mathfrak{D}$. Hence, we could equally write the link load constraint (7) for each extreme point of $\mathfrak{D}$, which are in finite but exponential number.

For a given routing f, the motivation in the traffic matrix generation step is to enumerate the extreme points of $\mathfrak{D}$ which correspond to the 'most challenging' traffic demands in terms of the arc utilization or the routing cost. Since $\mathfrak{D}$ is a polyhedral set, the algorithm will terminate after a finite number of iterations. Besides, the greedy choice of extreme points would lead to much fewer iterations before termination. We provide the pseudo codes of two different strategies in Algorithm 1 and Algorithm 2, respectively.

The first step *INITIALIZE* and the final step *CHALLENGE* are common for the two strategies. To start, we need an initial $d_0 \in \mathfrak{D}$.

Let $\mathfrak{D} = \{d \in \mathbb{R}^{|Q|} : Ad \leq \alpha, d \geq 0\}$ be the polytope of feasible traffic matrices with $A \in \mathbb{R}^{K|Q|}$ and $\alpha \in \mathbb{R}^K$. Then (7) implies that

$$l_a \geq \max_{d\in\mathfrak{D}} \sum_{(s,t)\in Q} d_{st} f_a^{st} \qquad a \in A. \tag{8}$$

To create d_0, in *INITIALIZE*, we solve maximization problem (8) for an arbitrary arc $a \in A$ by setting $f_a^{st} = \alpha$ for all commodities $(s,t) \in Q$, where α can be any positive constant. Then we create $\tilde{D}$ to hold all the traffic matrices that we generate throughout the algorithm. On the other hand, the aim of the step *CHALLENGE* is to determine a 'challenge' case to compare the routing cost and the maximum link utilization of the two routings. For this purpose, we take $d^{max} = \text{argmax}_{d\in\mathfrak{D}} \sum_{(s,t)\in Q} d_{st}$ as our challenge traffic matrix. Notice that we choose d^{max} independently of any performance measure or any topological information. At this stage, we are only interested in the traffic matrix that requires the utmost use of network resources. Although we could use some other criteria at this stage, we believe the current choice is fair enough since we will use d^{max} for comparison. Between *INITIALIZE* and *CHALLENGE*, there is the *MAIN* step, where the two strategies differentiate.

The first strategy is based on the greedy search of a new feasible traffic matrix based on total routing cost. Algorithm 1 outlines this strategy, which we call *CM* in the rest of the paper. At iteration *cnt* of

CM, we have an OSPF routing g^* with the minimum average routing cost $\Phi_{\tilde{D}}$ for the traffic matrices in $\tilde{D}$. The question we want to answer is: Does there exist another demand $d \in \mathfrak{D} \setminus \tilde{D}$ that costs more than $\Phi_{\tilde{D}}$, if we route it using g^*? To tackle this question, we first solve the following MIP model ($P_{MaxCost}$):

$$
\begin{aligned}
& \max \sum_{a \in A} \phi_a & & \\
\text{s.t.} \quad & \phi_a - u_z l_a \leq M(1 - y_a^z) - v_z c_a & & a \in A,\ z \in Z \\
& \sum_{z \in Z} y_a^z = 1 & & a \in A \\
& l_a - \sum_{(s,t) \in Q} d_{st} g_a^{st^*} = 0 & & a \in A \\
& \sum_{(s,t) \in Q} a_k^{st} d_{st} \leq \alpha_k & & k = 1, .., K \\
& \phi_a,\ l_a \geq 0 & & a \in A \\
& y_a^z \in \{0, 1\} & & a \in A,\ z \in Z \\
& d_{st} \geq 0 & & (s,t) \in Q
\end{aligned}
\tag{9}
$$

where y_a variables show the segment of the objective function that each ϕ_a lies in and (9) ensures that we obtain a feasible traffic matrix $d^{new} \in \mathfrak{D}$. Notice that g^* is not a variable anymore in $P_{MaxCost}$. Thus, the link load l_a is defined as a linear function of the demand variables $d \in \mathfrak{D}$ with coefficients $g_a^{st^*}$ obtained in the most recent iteration of the local search procedure of IGP-WO. In consequence, the solution of $P_{MaxCost}$ will be the worst case traffic matrix d^{new} leading to the highest routing cost $\sum_{a \in A} \phi_a^*$ for g^*.

Since $\mathfrak{D}$ is nonempty, $P_{MaxCost}$ will always yield a feasible traffic matrix d^{new}. However, there is no guarantee that we will get a new $d^{new} \notin \tilde{D}$ at each iteration since (9) ensures $d^{new} \in \mathfrak{D}$ but not $d^{new} \in \mathfrak{D} \setminus \tilde{D}$. To shun fake updates, we keep track of all matrices in $\tilde{D}$ using a hashing table. This is similar to what we use to avoid cycling in the tabu search algorithm for optimizing link weights. In brief, we use a hashing function to map each d^{new} to an integer $h_{d^{new}}$ and we mark its generation in the $h_{d^{new}}$ entry of a boolean table. Each time we solve $P_{MaxCost}$, we decide whether or not we should update $\tilde{D}$ using the boolean table and continue with the next iteration only if we have a new traffic matrix $d^{new} \notin \tilde{D}$ for which the routing cost $\sum_{a \in A} \phi_a^*$ is higher than the current average cost $\Phi_{\tilde{D}}$.

On the other hand, the second strategy is greedy in the sense of traffic load on arcs. It uses link utilization as the determining factor for new traffic matrix generation. Basically, given an OSPF routing g^* optimal for $\tilde{D}$, it looks for a traffic matrix d^{new}, which makes some arc $a \in A$ overloaded or increases the current congestion rate of the network, that is $U^{max} = max_{a \in A, d \in \mathfrak{D}} \frac{l_a}{c_a}$. We use the hashing function that we have described above to keep track of the traffic matrices in $\tilde{D}$ and avoid cycling. A framework of this strategy is provided in Algorithm 2. We will refer this strategy as *LM* from now on.

The main difference between *CM* and *LM* is the domain of the challenge. *CM* generates a demand matrix d^* that puts the network in a worse situation as a whole for a given routing configuration on the basis of the total routing cost. On the contrary, in *LM*, the new traffic matrix is at least 'locally' challenging, since we consider the worst case for each arc individually. In both strategies, we enumerate at most one traffic matrix at each iteration. However, we can modify Algorithm 2 to generate multiple traffic matrices, namely at most one for each arc. Finally, each time the algorithm performs a tabu search, it starts with the optimal weight metric of the most recent iteration. This is useful to reduce the time spent for re-optimizing the weight metric in the *IGP-WO* stage.

Algorithm 1 Strategy 1 with Cost Maximization - *CM*

Require: directed graph $G = (V, A)$, traffic polytope $\mathfrak{D}$, link capacity vector c;
Ensure: minimum cost OSPF routing f^* and metric ω^* for $(G, \mathfrak{D}, c)$;

INITIALIZE:
  Find an initial feasible demand matrix $d_0 \in \mathfrak{D}$;
  $d^{rec} \leftarrow d_0$; // d^{rec} : the most recently enumerated demand matrix;
  $\tilde{D} \leftarrow d_0$; // $\tilde{D}$: current set of demand matrices enumerated so far;
  $NewDem \leftarrow TRUE$;
  $cnt = 0$;
MAIN:
while $(cnt \leq cnt\_limit)$ and $(NewDem = TRUE)$ **do**
  IGP-WO: Find an optimized oblivious OSPF routing g^* for $\tilde{D}$ and the associated metric ω_T^*;
  Get $\Phi_{\tilde{D}}$: the average routing cost for $\tilde{D}$;
  $NewDem = FALSE$;
  Solve $P_{MaxCost}$ to get $\sum_{a \in A} \phi_a^*$ and d^{new};
  if $\sum_{a \in A} \phi_a^* > \Phi_{\tilde{D}}$ and $d^{new} \notin \tilde{D}$ **then**
    $\tilde{D} \leftarrow d^{new}$;
    $NewDem = TRUE$;
    $cnt \leftarrow cnt + 1$;
  end if
end while
$f^* \leftarrow g^*$;
$\omega^* \leftarrow \omega_T^*$;
CHALLENGE:
  Find the challenge demand matrix $d^{max} = \text{argmax}_{d \in \mathfrak{D}} \sum_{(s,t) \in Q} d_{st}$;
  Get $\Phi_{d^{max}}^*$ // the cost of routing d^{max} with f^*;
  Get $U_{d^{max}}^*$ // the congestion rate for d^{max} with f^*;

Algorithm 2 Strategy 2 with Arc Load Maximization - *LM*

Require: directed graph $G = (V, A)$, traffic polytope $\mathfrak{D}$, link capacity vector c;
Ensure: minimum cost OSPF routing f^* and metric ω^* for $(G, \mathfrak{D}, c)$;

```
INITIALIZE // As in Algorithm 1
MAIN:
while (cnt ≤ cnt_limit) and (NewDem = TRUE) do
  IGP-WO: Find an optimized oblivious OSPF routing g* for D̃ and the associated metric ω*_T;
  U^max = maximum link utilization for d^rec;
  NewDem = FALSE;
  a = 0 // start with the first arc of G;
  while (a < |A|) and (NewDem = FALSE) do
    d^new = argmax_{d∈D}(g*_a d); // d^new : worst case demand matrix for a with routing g*;
    if (g*_a d^new > c_a) or ((g*_a d^new)/c_a > U^max) then
      if d^new ∉ D̃ then
        d^rec = d^new;
        D̃ ← d^rec;
        NewDem = TRUE;
        cnt ← cnt + 1;
      end if
    end if
    if NewDem = FALSE then
      a ← a + 1;
    end if
  end while
  if NewDem = TRUE then
    cnt ← cnt + 1;
  end if
end while
f* ← g*;
ω* ← ω*_T;
CHALLENGE //As in Algorithm 1
```

4 Conclusion

We have surveyed meta-heuristics for traffic engineering in the framework of intra-domain routing. The available techniques cover the basic problem and some important extensions that seem sufficient for the needs of most network operators.

Nevertheless, there is still room for improvement. In particular, understanding the interactions between inter-domain and intra-domain routing and developing tools that allow for inter-domain or joined inter/intra-domain optimisation remains a big challenge.

Acknowledgments

This work has been partially supported by the Walloon Region (DGTRE) in the framework of the TOTEM project, and the Communauté française de Belgique - Actions de Recherche Concertées (ARC).

References

[1] A. Altın, P. Belotti, and M.Ç. Pinar. OSPF routing with optimal oblivious performance ratio under polyhedral demand uncertainty. Technical report, Bilkent University, 2006.

[2] A. Altin, B. Fortz, M. Thorup, and H. Ümit. Intra-domain traffic engineering with shortest path routing protocols. *4OR*, 7(4):301–335, 2009.

[3] A. Altin, B. Fortz, and H. Ümit. Oblivious OSPF routing with weight optimization under polyhedral demand uncertainty. Technical Report 588, ULB Computer Science Department, 2008. http://www.ulb.ac.be//di/publications/RT_2008.html.

[4] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 313–324, New York, NY, USA, 2003. ACM.

[5] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6:154–160, 1994.

[6] P. Belotti and M.Ç. Pinar. Optimal oblivious routing under statistical uncertainty. *Optimization and Engineering*, 9(3):257–271, 2008.

[7] W. Ben-Ameur, E. Gourdin, B. Liau, and N. Michel. Optimizing administrative weights for efficient single-path routing. In *Proceedings of Networks 2000*, 2000.

[8] W. Ben-Ameur and H. Kerivin. Routing of uncertain demands. *Optimization and Engineering*, 3:283–313, 2005.

[9] A. Bley. A Lagrangian approach for integrated network design and routing in ip networks. In *Proceedings of the 1^{s}t International Network Optimization Conference (INOC 2003), Paris*, pages 107–113, 2003.

[10] L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46(1):36–56, 2005.

[11] Luciana S. Buriol, Mauricio G. C. Resende, and Mikkel Thorup. Speeding up dynamic shortest-path algorithms. *INFORMS J. on Computing*, 20(2):191–204, 2008.

[12] M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. Combinatorial Optimization*, 6:299–333, 2002.

[13] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40(10):118–124, 2002.

[14] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *Proc. 19th IEEE Conf. on Computer Communications (INFOCOM)*, pages 519–528, Tel-Aviv, 2000.

[15] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, 2002.

[16] B. Fortz and M. Thorup. Robust optimization of OSPF/IS-IS weights. In W. Ben-Ameur and A. Petrowski, editors, *Proc. INOC 2003*, pages 225–230, Paris, October 2003.

[17] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1):13–48, 2004.

[18] B. Fortz and H. Ümit. Efficient techniques and tools for intra-domain traffic engineering. Technical Report 583, ULB Computer Science Department, 2007. To appear in *International transactions in Operational Research*.

[19] K. Holmberg and D. Yuan. A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48:461–481, 2000.

[20] Cisco Systems Inc. *Internetworking Technologies Handbook, Third Edition*. Cisco Press, 2000.

[21] G. Leduc, H. Abrahamsson, S. Balon, S. Bessler, M. D'Arienzo, O. Delcourt, J. Domingo-Pascual, S. Cerav-Erbas, I. Gojmerac, X. Masip, A. Pescaph, B. Quoitin, S.F. Romano, E. Salvatori, F. Skivée, H.T. Tran, S. Uhlig, and H. Ümit. An Open Source Traffic Engineering Toolbox. *Computer Communications*, 29(5):593–610, March 2006.

[22] J. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.

[23] E. Mulyana and U. Killat. An alternative genetic algorithm to optimize OSPF weights. In *15th ITC Specialist Seminar*, pages 186–192, Würzburg, 2002.

[24] E. Mulyana and U Killat. Optimizing IP networks for uncertain demands using outbound traffic constraints. In *Proc. INOC 2005*, pages 695–701, Lisbon, 2005.

[25] M. Pióro and D. Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufman, 2004.

[26] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.